SIDNEY W. KITCHEL*

Manager of Database Development
Data-Parallel Systems, Inc.
Bloomington, Indiana

Object-Oriented Databases for Libraries and Other Complex Systems

INTRODUCTION

If the author had been as creative as Debora Shaw (these proceedings), the title of this paper could have been "Libraries and Object-Oriented Database Systems, or When Do You Need an SST?" However clever the title, the intent of the following discussion is to examine some of the latest developments in database technology and to conjecture how they might be applied to information processing within the library world. The main new development that will be considered is an object-oriented database system. But other new developments will be addressed as well.

Database systems and practice have developed to satisfy an organization's critical needs for operational data. A database management system (DBMS) is supposed to make it easy to share and protect vital data and information. The designers of such systems are charged to get all the right information into the system, make it easy for multiple sub-organizations to get at it and, yet, prevent the wrong eyes from seeing the parts of it they have no right to. DBMSs are further charged to make sure that a minimum of crucial information is ever lost by accident or disaster, or is destroyed by miscreants.

Libraries are certainly organizations very dependent on information. In fact, the case could be made that information is more their lifeblood than it is for airlines or banks. Libraries exist to provide information to patrons in the form of books and other media, direct services of

*Formerly Senior Software Engineer, Dome Software Corp., Carmel, Indiana, which was formerly Scientific Software Products, Inc.

reference specialists and, more recently, electronic delivery of information. However, the organization's operational information in databases, as opposed to self-descriptive data about the database itself, is always about something else. It is about a seat reserved on flight 507 by S. Jones on June 23. Or it is about Acme Corporation's loan that is due on the 25th of July. Libraries also need to store information about their activities. Accessions, cataloging, acquisitions, holdings, and circulation data all need to be recorded and managed. This is information that runs the library and that is not usually completely available to patrons.

So in the case of libraries, database systems perform at multiple levels of activity. They are used by local libraries to hold and process operational data. They are national depositories of both operational data, as in OCLC holding bibliographic data required for cataloging, and pure information as in the Dialog or Nexis services. All these observations lead to the conclusion that effective databases are crucial to a library's successful daily operation. Therefore, it is safe to assume that many systems are already in place and serving their libraries well. Who could ask for anything more? Well, most systems in place could be improved and some systems are only barely keeping their heads above water. So what are the technical problems facing these systems?

PROBLEMS OF LIBRARY DATA PROCESSING

Any organization doing information processing encounters difficulties from time to time. Often they are products of outside pressure on the organization to perform. But internal circumstances may also dictate the level of data processing happiness. The three severest problems that face designers of data processing and database systems applied to the library world are:

1. extremely sophisticated search of large groups of sources required;
2. high-volume transaction processing (Lynch, 1987); and
3. complex entities to model (Bancilon, 1988).

Clearly, not every application faces all three of the problems simultaneously. Many systems often are under stress from patron demands alone. And of course, some systems are still composed of manual card files and people. However, one of the three problems is faced by almost all large institutions. Some organizations such as the bibliographic utilities may confront all three simultaneously. Let us look at each problem separately.

Search in conventional database systems is fairly benign and simple: To find all the payments under contract number CN82723, turn to your computer terminal and request a report keyed on the contract number. Shortly on your screen or the printer, the list appears. A library patron's requests for information are often not so simple. A typical search might be to look for all recent books and articles about object orientation. The reference librarian cannot just turn to a terminal and type a query. First of all, there is no single system that covers all published materials. So expert knowledge is applied just to determine which bibliographic resources to consult. But once a set of them is chosen, it is still a difficult task to form good search strategies for each source. Does one search for "object orientation" or "object" or possibly "object-oriented"? Probably, the reference librarian must further question the patron for more details of the real subject desired, which turns out to be recently offered, commercial, object-oriented databases in the U.S. Being sure that what is retrieved is exactly what is sought, and that the final search result is complete is most difficult and, all too often, most expensive.

Needing to complete huge numbers of transactions per day is a real problem for library service organizations. Shaw's (these proceedings) survey of library database applications (in these proceedings) shows a reflection of this problem. She tabulates characteristics of ten large systems and none of them use a relational DBMS. Commercial relational systems provide adequate tools to meet many business needs. They, however, do not provide high rates of transactions. They are fairly simple to construct and use, but are still too slow for high-volume operation. Libraries and businesses are left to either "roll their own" or adopt systems using older technology but offering a large transaction rate. In the future, the newer database systems will mature and offer their owners better help with numerous transactions.

Complexity is a problem in and out of the library world. The auto engineer sitting at a high-powered workstation would like to keep his car design ideas well organized. A car has hundreds of subassemblies and thousands of individual parts. The design engineer may have several versions of a design for any of the subassemblies or parts. It is a fairly difficult task to get a database system to store all the versions and all the data that describes the car design. Librarians also face relatively complex entities that must be described in databases. The MARC format itself is complex and, thus, the storage of bibliographic information is no simple task, especially if there are added constraints of efficient storage and rapid retrieval. Conventional database systems groan at being bent to achieve the modeling, storage and manipulation of such complex objects.

Complex entities and complex relationships are simple problems for an object-oriented database system (OODBS). It is specifically designed to handle complexity more easily than relational systems can. The next section describes the nature of OODBSs and how they deal with complexity.

Object Orientation

To understand the promise of the new technology inherent in an OODBS, a few basic concepts must be grasped (Peterson, 1987; Shriver & Wegner, 1988). These concepts may be understood by answering some initial questions. First, what are objects, anyway, and how do they behave? Second, how can objects fit into a database system? Third, how does one model one's world with an object-oriented system? And last, what are the implications of object orientation for system development and maintenance?

An *object* in an object-oriented system is the computer reflection of a real world object (see Figure 1). This is especially true of database objects, whose job is to model a thing in the world that one wishes to keep track of. Computer objects, like their counterparts, persist and have a unique identity, which means no matter how much they look alike, they are always distinct. Real world objects are like this, too: take two new baseballs off the shelf at a sporting goods store and they are distinct, even though they look and measure the same.

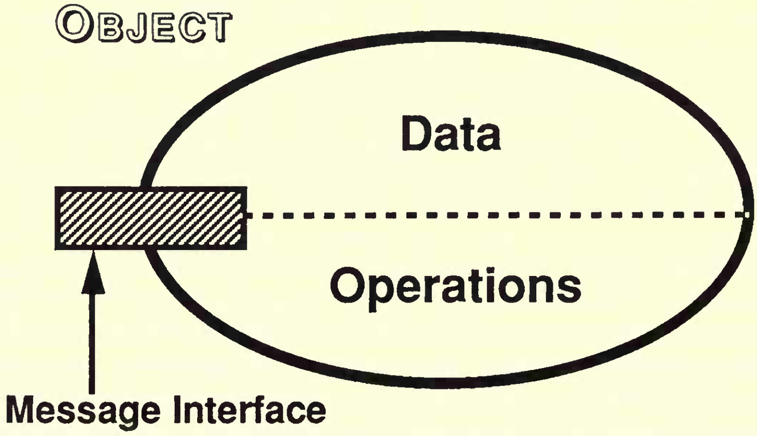

Figure 1. An object

An early commercial OODBS was Vbase, for which there are a whole series of manuals and guides (Ontologic, Inc., 1987). In addition, Hewlett-Packard is working on a commercial OODBS called Iris DBMS

(Fishman et al., 1987). Dome Software offers a networked system of workstations and an OODBS server running on a DEC VAX called DOME (Distributed Object Management Environment) but previously called LCE (Kachhawaha & Hogan, 1987); Servio Logic offers an OODBS called GemStone (Maier et al., 1986); and both Servio Logic and Park Place Systems, an offshoot from Xerox's Palo Alto Research Center, have tried to build an OODBS by making objects created while running the programming language SmallTalk persistently over long periods (Copeland & Maier, 1984).

A computer object is a tool for encapsulation of data and operations. The data is used to record things about the real object, while the operations may tell the outside world about the data or manipulate it. For example, let us go back to the design engineer keeping track of his new car. One object that might be in the OODBS for car design is a door. The designer could store information about materials to make the door and its physical characteristics such as measurements. To aid in viewing the object, the system might well contain operations to display the data both as numbers and to draw the door on a bit-mapped screen. So an operation might be a "read" that would get the numbers in the data section of the object, but could also have operations like "draw" or "rotate" as well.

Because encapsulation builds a wall around the data, a user can only ask questions about them and request actions be taken on them through the operations. A computer scientist describes what we have so far as an "abstract data type." Object orientation actually simplifies the usual concept of an abstract data type. In an object-oriented system, the interaction with objects comes through messages to objects. Messages allow a uniform interface to all objects, because each message has the same form and any message may be sent to any object. Of course, if the object cannot understand the message, it will just reply that it does not know the operation the user wanted. A normal abstract data type has a more complicated interface made up of all the public operations on the data type, which is not regular. To communicate one must know the exact rules of these operations. Objects are simpler, because a message always has the form:

target-object selector {optional parameters}

Here the "target-object" must be the name of some existing object, "selector" is the name of an operation, which is usually called a method, and the "optional parameters" are any data that the method needs to work on. For example, in an object-oriented graphics package, a typical message might be:

```
triangle(T23)
     rotateRight 105,
```

which would ask the triangle named "T23" to rotate itself to the right by 105 degrees. We could also send the message:

```
user(Sally)
    rotateRight 47.
```

Chances are, the user objects are in the system just to identify people who can log on and an instance of the user class cannot rotate. This is an example of the uniform interface—even though a user cannot rotate itself, the message is still legal, is delivered, but gets the reply that "rotateRight is not a selector for objects of the class user."

Objects group together to form classes. A class is both a container of all like objects and a way to make a definition of an object. In fact, an object comes into existence when a class object is sent the message **triangle new**. This asks the class object "triangle" to run the method that creates a new instance of the class. Having classes allows a group of objects to be organized by classifications in a class hierarchy. In an application to manage a zoo, we might see a class hierarchy, as in Figure 2. The class hierarchy establishes an **IS A** relationship. A lion is a cat. The class hierarchy allows a new property of objects—they inherit the characteristics of their ancestor classes. Thus, if a cat object holds a field or part "HairColor" to record the animal's color, then the definition of the lion class would not need such a field because it would be inherited from the cat class definition. Similarly, any methods that are defined for an ancestor are inherited by a subclass. So, if there were a method that looked at a whole class of birds, checking each bird instance for eggs to average the number of eggs laid per season, it would be automatically available to the Eagle class and the Hawk class. The developer would have to expend no effort to add it. Inheritance is a powerful tool for organizing a data model and for reducing unwanted duplication of effort.

In a library OODBS, there might be classes that represented bibliographic entities, patrons, and services such as being at the bindery. Imagining that the system is for a state university library, the following pieces of the class hierarchy might exist: classes for patrons (see Figures 2 and 3) and classes for bibliographic records (see Figure 4). So far, objects have been said to contain data, but the nature of it has not been related. Figure 5 shows a little of what an instance of the patron class might contain. It has some simple data such as strings for parts such as names, but there can be more complicated items. Some data is convenient to aggregate. In the example, an address is a part of a patron object, but it is an owned instance of an address class. Being an owned instance allows the patron's address both to act as a "chunk" of information with its own name "PatronAddr" and to have internal structure that may be used in searching or organizing reports on patrons. The next parts are even more interesting. The "Dept" part is a reference to a separate or shared instance of the department class. Clearly, an individual patron should not own the information about a university department. That information should stand alone. Aggregation and reference are examples of relationships between objects that are strongly coupled or weakly coupled. If patron Sally Jones is in the Economics Department, deleting her object should not remove any information inherent to the department, but it would be quite all right to also automatically delete her owned address object. It only pertains to Sally, while other objects may well wish to relate themselves to the Economics object by reference.

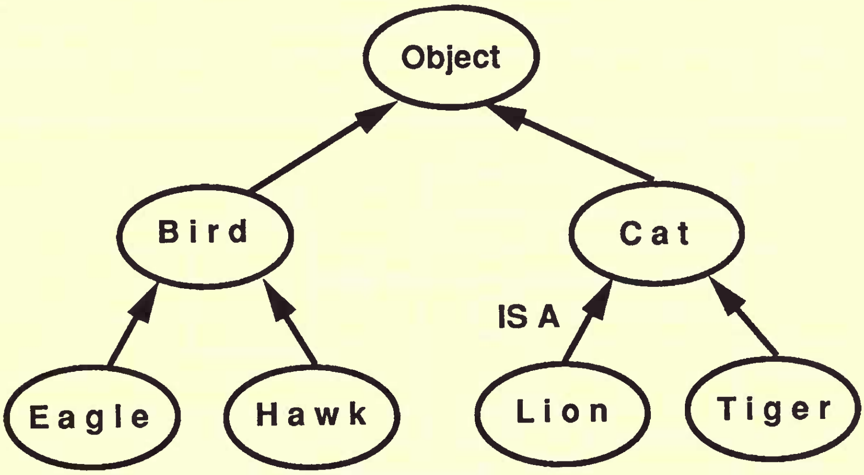

Figure 2. Class hierarchy

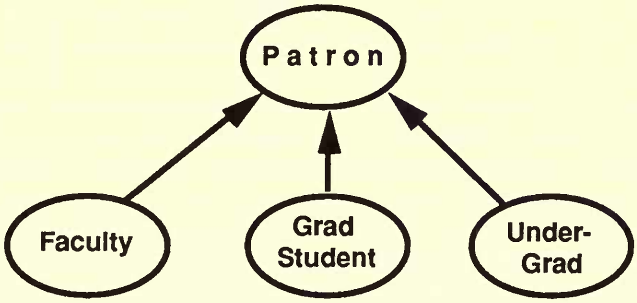

Figure 3. Patron subtree

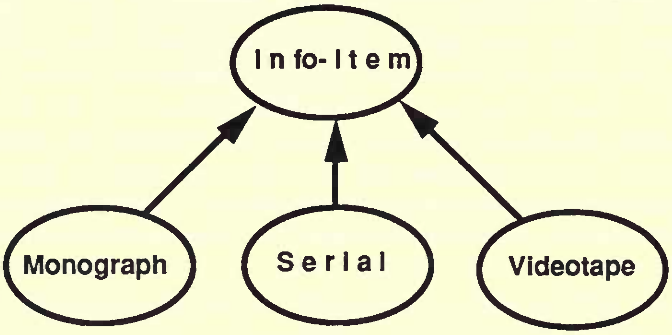

Figure 4. Bibliographic classes

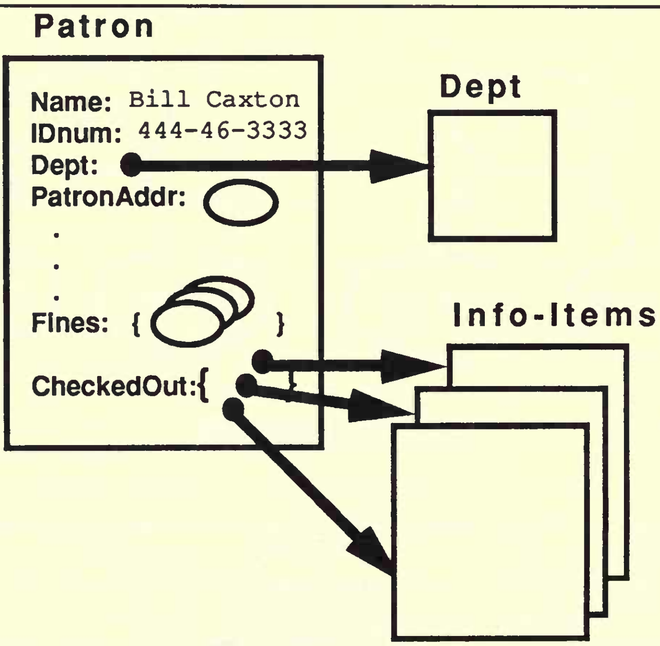

Figure 5. Patron class definition

Another useful kind of part is a set or collection, which allows an element of an object to have variable size. A set may contain either owned instances of a class or references. In the library example, the part "CheckedOut" is a set of references to material that the patron has checked out. This part can contain zero or more references to the

Info-Item class of bibliographic information. Each instance of the Info-Item class corresponds to a shelflist card in a manual library system. A sophisticated OODBS would allow a search for the objects that reference an Info-Item. So the checkout method could check to make sure no one has this particular item currently checked out. The "Fines" part is an example of a set of owned objects. It would contain instances of a class that recorded fines that are due. If the Patron instance "Bill Caxton" is deleted from the system, all information about his fines would disappear, but information about his department or books he might have checked out will not be lost.

The above example shows the basic tools that OODBSs offer for modeling the real world. An object may contain simple data, single aggregations, or collections of aggregations. It may be related to another independent object by a reference as a single part or to many other objects by a collection of references. All these tools give the developer of an OODBS the ability to create a design that more naturally captures the relationships found in the organization. It gives the developer a much better chance to get the design right. A developer using a relational tool must discover the same real relationships, but is faced with the more perilous task of translating the natural model of the system into a set of relational tables. This must be done very carefully to avoid nasty design faults.

Traditional computer systems built around databases have always had two main divisions. The database has been analyzed and designed looking at the needs for long-term data storage and for processing. Separately, a group of application programs is written to actually do the needed processing. Thus, traditional systems have always made a strong separation of data and operations. The result has been extremely high maintenance costs. Every time an error in the original design is found, or when the users want an alteration in that design, massive changes must be made. Some systems have hundreds of programs that depend on the database design to work properly. Change the structure of the database and many or all of the programs must be altered to work. In an object-oriented system, much of this change can be avoided. In an OODBS, the code to manipulate data stays with the data. It is in one place and is easier to maintain. If the structure of data in an object changes, say, by having a new part added, most application programs can completely ignore that structural change. They depend on messages to get data from an object. Once the object has changed, the old messages will behave the same, if they are specific. For example, if the message was to a patron instance asking for the patron's address, it will still work even if a new part, birthdate, has been added after "IDnum." What object orientation does is hide the details of actual storage and manipulation, so no program need depend on them. There is no longer any public physical database design—it is a private matter known only to the system designers. What is public is the existence of the data and how to get at it.

Object orientation for databases offers much promise to system designers and users alike. It will increase the productivity of the computing staff so that they may get more work done and have it be more effective and reduce the errors. Thus, it will improve the overall quality of the library's software. In the long term, OODBSs should be easier to maintain and cost less. Being newer, they also come along with better tools for user access as well. They are more likely to have fancier and friendlier interfaces, built on visual metaphors that users will find easier to like and use. The developer will be given tools to create tailored interfaces that meet the exact needs and skills of the end users.

Other Database Research Directions

Database researchers have not all been working at object orientation. Research has also taken other directions for its explorations. This section outlines some likely areas of work in the near future.

An issue that is a central facet of OODBSs is the marriage of data and operations. Some database researchers are not interested in accomplishing a union. They prefer to concentrate on the data modeling possibilities of systems that are richer than the relational paradigm, but leave the operations out. Most of this sort of research falls under the heading of the nested relational model, which provides a relational system in which the restriction of allowing only simple or atomic values in the columns of a table is relaxed. A relation with only atomic values is said to be in **first normal form** and, thus, a nested relational table is said to be a non-first normal form relation. The nested relational approach, like the object-oriented one, permits a designer to capture more naturally the complexity of a real-world problem domain. When sets of some entity occur, for example, when a patron has multiple fines, the table holding data about the person may have a column that contains a subtable of all the fine data. In a traditional relational system, a second table would have to be created and a data relationship on some unique identifier established to bridge the person table and the fine record table. Retrieving the natural set of fines would require a join operation that can be expensive. In the nested relational version, having a person's record would automatically provide access to all the fines in the subtable. This is also true of an object-oriented version in which a set of fine objects would be inside the person object.

The ability to nest a table or to include a set of objects exactly parallels what a hierarchical database could do. However, current database researchers would probably describe these older, hierarchical, and network databases as obsolete. They are out of date because they lack the firm theoretical foundation of relational systems, they have inflexible user interfaces and rigid physical storage structures, and they are less maintainable than an OODBS.

A major selling point of relational systems is their flexibility. A database built on a relational foundation does not predetermine the relationships between data elements. The system is able to exploit all the relationships actually in the data. The lack of this flexibility is the subject of significant criticism of the object-oriented model by the advocates of the relational model. But the relational researchers are not idle; they see ways to expand the usefulness of their systems into a "post-relational model." The central aspect of post-relational systems is extensibility. They, like the OODBSs, will offer the database designer better data modeling by allowing nested relations (these researchers are concerned with storing and manipulating complex entities such as graphics), better management of queries by allowing them to be stored and compiled, and better overall behavior by allowing procedures to be stored in the database. The designer will also have more control over the structure and internal working of the implemented system. The post-relational researchers hope to provide a significant leap in power to the designers so that each system will perform near its optimum expected behavior (Stonebreaker & Rowe, 1986; Carey & DeWitt, 1985; Lindsay, 1987).

A related research area is the work being done to put more intelligence into database systems. Some of this work is a bit blurred with the extension of relational systems, because one possible way to extend a relational database system is to give it intelligence. However, some of the basic research in this area predates the call for relational extension. There are several components to this research. The oldest is the desire to improve the query interface of existing relational systems, which usually means combining a logic programming language such as PROLOG with a database. The basic idea here is to give more power to query writers. A second area of research in this subfield examines how to combine expert systems with databases. Expert systems are a commercial outgrowth of research in artificial intelligence. They attempt to capture the knowledge of expert humans in a computer-based system that can perform at levels near the human expert. They usually involve a store of expert experience in a knowledge-base and an inference engine for handling outside demands by processing the knowledge base. Expert systems have been built to do medical diagnosis, repair diesel engines, and plan the installation of all new VAX computers. Combining expert systems with regular databases appears to be a quite fruitful area for future development.

Yet another area of research is putting knowledge and rules into databases. In an unsophisticated fashion, this has been going on for some time in the form of data entry checking. The central idea is to make the database system more knowledgeable about the world it lives in. So besides just holding the data for a business, the system would hold rules about that world. For example, if the system were in a library, it would be told the rules of catalog numbers. When a book is cataloged, it would warn of an incorrect call number that is being created in the shelflist database. Or it would know that most undergraduates cannot carry more than twenty or thirty books and would object to a single person attempting to check out that many. The database system would be given rules about how a library operates that would constitute meta-data about the library world. Meta-data allows a database system to work more reasonably with fewer errors and to help monitor all activity that is captured by information stored in the database (Brodie, 1988; Hallaire, 1981; Ceri et al., 1986).

Once database systems have been extended and smartened, wouldn't they be expected to start doing things on their own? Some database researchers do expect it. They are looking forward to "active databases" that will have a rule knowledge base that gives them the meta-data about their environment. So when some event occurs or some set of predetermined circumstances comes about, the active database will do something. In the library system mentioned above, if a book that is "popular" according to check-out statistics is removed from the shelflist, the database could automatically generate a purchase recommendation that would be brought to the attention of the acquisitions staff. The active database will send a mail message to an acquisitions librarian and enter the recommendation in a special log. This is an extension of an older database idea called *triggers*. Events or data coming through cause the system to take some action. Often it is to do some logging or bookkeeping that is required. But once procedures are put into an extended database system, the active database can do anything the computer can do (Dayal et al., 1988).

CONCLUSION

In a number of ways, the library world has been a leader in what is popularly referred to as the information revolution. But because of limited resources, it is not likely that libraries and librarians can continue to lead the revolution. Limited budgets just do not permit it. However, it seems clear that there are two figurative Bastilles that must be stormed before the revolution can be won.

The first is technological. Many librarians seem to think that installing a personal computer, even one with CD-ROM databases, is a great step forward towards winning the fight. To this author's view, putting something as primitive as a DOS-based machine into the hands of patrons is not necessarily a step forward. The technological gap must be bridged to win the revolution. The gap is not just the absence of computers in libraries; it is also the existence of obsolete and hard-to-use computers getting in the patron's way. The Bastille that must be stormed is unfriendly machines and programs. Easy (meaning REALLY easy here, so that the computer illiterate off the street can use it) and intelligent access to the information is a must (Baecker & Buxton, 1987; Barstow et al., 1984; Smith & Green, 1980; Hartson & Hix, 1989). What is the good of a revolution if only a handful of citizens may partake? To win an information revolution, access to information must be made as easy as using the manual card catalog. The tools provided will have to be intelligent, so they will have to be based on systems that are built by knowledge engineers using the best fruits of artificial intelligence research. The technology advances needed are quite significant. In many areas, even a start has not been made and, where a beginning exists, there is often frustration with the difficulty of the task ahead (Blair & Maron, 1985; Witten & Bramwell, 1985).

The second area needing aggressive attention is economical. What good is access to billions of pieces of data and the use of fancy computer and AI tools, if the patron can't afford to use them? Presently, any area librarian can search a national citation resource on Dialog and in less than five minutes can find hundreds of citations that might be useful in researching a patron's paper. But the patron can't afford the service because searching Dialog databases for a few minutes can cost hundreds of dollars. There can be no true information revolution until the average citizen can have affordable access to the information needed. It must be as affordable as books are in present day libraries. For the near future, this Bastille is even tougher than the technological one. It is fair to say that achieving the goal of easy access will mean increasing the revolution's cost, because it will take fancier equipment and much more thinking and programming.

Object-oriented technology would be quite beneficial in many applications in the library world. Systems built around an OODBS would be very cost-effective. OODBSs could help support the technological advances needed to bring about a true information revolution. However, will we see these advances soon? This is very much like the question, "When do you need the SST?" Remember that when that question was asked in the United States, citizens answered, "Not now, maybe not ever; we can't afford it." When the question was asked in Europe, they answered, "Well, we'd like to build the SST like the Americans envisioned, but we don't have the technology to do it, so we'll build a Concorde instead and act like we built the SST." (Recall that the Concorde flies about 500 mph slower than the planned SST and carries about 100 fewer passengers.) One guess is that, for a while, the answer to "Do we need OODBSs in libraries?" will be "Not now, maybe later."

REFERENCES

Baecker, R. M., & Buxton, W. A. S. (Eds.) (1987). *Readings in human-computer interaction: A multidisciplinary approach.* Los Altos, CA: Morgan Kaufmann.

Bancilon, F. (1988). Object-oriented database systems. In *Proceedings of the seventh ACM SIGACT-SIGMOD-SIGART symposium on principles of database systems* (21-23 March, 1988) (pp. 152-62). Austin, TX: Association for Computing Machinery.

Barstow, D. R.; Shrobe, H. E.; & Sandewall, E. (Eds.) (1984). *Interactive programming environments.* New York: McGraw-Hill.

Blair, D. C., & Maron, M. E. (1985). An evaluation of retrieval effectiveness for a full-text document-retrieval system. *Communications of the ACM, 28*(3), 289-299.

Brodie, M. (1988). Future intelligent information systems: AI and database technologies working together. In J. Mylopoulos and M. Brodie (Eds.), *Readings in artificial intelligence and databases.* San Mateo, CA: Morgan Kaufmann.

Carey, M. J., & DeWitt, D. J. (1986). Extensible database systems. In M. L. Brodie & J. Mylopoulos (Eds.), *On knowledge base management systems: Integrating artificial intelligence and database technologies* (pp. 315-30). New York: Springer-Verlag.

Ceri, S.; Gottlob, G.; & Wiederhod, G. (1986). Interfacing relational databases and Prolog efficiently. In L. Kerschberg (Ed.), *Proceedings of the 1st International Conference on Expert Database Systems* (Papers presented at the 1st International Conference, Charleston, South Carolina, April 1986) (pp. 141-53). Menlo Park, CA: Benjamin/Cummings.

Copeland, G., & Maier, D. (1984). Making SmallTalk a database system. In *Proceedings of the 1984 ACM SIGMOD Conference on Management of Data* (Boston, MA, 18-21 June 1984) (pp. 316-25). New York: Association for Computing Machinery.

Dayal, U.; Buchmann, A. P.; & McCarthy, R. M. (1988). Rules are objects too—A knowledge model for an active, object-oriented database system. *Lecture Notes in Computer Science, 334,* 129-143.

Fishman, D. H.; Beech, D. H.; Cate, H. P; Chow, E. C.; Connors, T.; Davis, J. W.; Derrett, N.; Hoch, C. G.; Kent, W.; Lyngbaek, P.; Mahbad, B.; Neimat, M. A.; Ryan, T. A.; & Shan, M. C. (1987). Iris: An object-oriented database management system. *ACM Transactions on Office Information Systems, 5*(1), 48-69.

Gallaire, H. (1981). Impacts of logic on databases. In *Proceedings of the 7th international conference on very large databases* (Cannes, France, 9-11 September 1981) (pp. 248-59). Washington, DC: IEEE Computer Society Press.

Hartson, R., & Hix, D. (1989). Human-computer interface development: Concepts and systems. *ACM Computing Surveys, 21*(1), 5-92.

Kachhawaha, P., & Hogan, R. (1987). LCE: An object-oriented DBMS for the research laboratory. *Spring DECUS U. S. Symposium 1987,* Nashville, TN, 27 April—1 May 1987) (pp. 43-55). Maynard, MA: Digital Equipment Users Society.

Lindsay, B. G. (1987). A data management extension architecture. In U. Dayal (Ed.), *Proceedings of the 1987 ACM SIGMOD Conference on Management of Data* (San Francisco, CA, May 1987) (pp.. 220-26). New York: Association for Computing Machinery.

Lynch, C. A. (1987). *Extending relational management systems for information retrieval applications.* Unpublished doctoral dissertation, University of California at Berkeley.

Maier, D.; Stein, J.; Otis, A.; & Purdy, A. (1986). Development of an object-oriented DBMS (OOPSLA 1986 Conference Proceedings). *ACM SIGPLAN Notices, 21*(11), 472-482.

Minker, J. (Ed.). (1978). *Logic and databases.* New York: Plenum Press.

Ontologic, Inc. (1987). *Vbase, integrated object system: Development tools guide for VAX/VMS.* Billerica, MA: Ontologic, Inc.

Peterson, G. E. (Ed.). (1987). *Tutorial: Object-oriented computing, volumes 1 and 2.* Washington, DC: IEEE Computer Society Press.

Shriver, B., & Wegner, P. (Eds.). (1987). *Research directions in object-oriented programming.* Cambridge, MA: MIT Press.

Smith, H. T., & Green, T. R. G. (Eds.). (1980). *Human interaction with computers.* London: Academic Press.

Stonebreaker, M., & Rowe, L. (1986). The design of POSTGRES. In C. Zaniolo (Ed.), *Proceedings of the 1986 ACM SIGMOD Conference on Management of Data* (Washington, DC, May 1986) (pp. 340-55). New York: Association for Computing Machinery.

Witten, I. H., & Bramwell, B. (1985). A system for interactive viewing of structured documents. *Communications of the ACM, 28*(3), 280-288.